



Índice

1. algorithm	1
2. Estructuras	2
2.1. RMQ (static)	2
2.2. RMQ (dynamic)	2
2.3. Fenwick Tree	2
2.4. Union Find	2
2.5. Disjoint Intervals	3
2.6. RMQ (2D)	3
2.7. Big Int	3
2.8. Modnum	5
2.9. Bittrie	5
3. Strings	5
3.1. Trie	5
3.2. Suffix Array	5
3.3. String Matching With Suffix Array	6
3.4. LCP (Longest Common Prefix)	6
3.5. Corasick	6

4. Geometría	7
4.1. Punto	7
4.2. Line	7
4.3. Segment	7
4.4. Rectangle	8
4.5. Polygon Area	8
4.6. Circle	8
4.7. Point in Poly	8
4.8. Convex Check CHECK	9
4.9. Convex Hull	9
4.10. Cut Polygon	9
4.11. Bresenham	9
4.12. Rotate Matrix	9
5. Math	9
5.1. GCD	9
5.2. LCM	10
5.3. Simpson	10
5.4. Fraction	10
5.5. Polinomio	10
6. Grafos	11
6.1. Dijkstra	11
6.2. Bellman-Ford	11
6.3. Floyd-Warshall	11
6.4. 2-SAT + Tarjan SCC	11
6.5. Articulation Points	12
6.6. LCA + Climb	12
7. Network Flow	12
7.1. Edmonds Karp's	12
7.2. Push-Relabel	13
8. Ayudamemoria	14

1. algorithm

#include <algorithm> #include <numeric>

Algo	Params	Funcion
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	void ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	void llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	it al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	bool esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	it encuentra i $\in [f, l)$ tq. i=elem, pred(i), i $\in [f2, l2)$
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2, l2) $\in [f, l)$
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	it min, max de [f, l]
lexicographical_compare	f1, l1, f2, l2	bool con [f1, l1] i [f2, l2]
next/prev_permutation	f, l	deja en [f, l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f, l), hace un heap de [f, l)
is_heap	f, l	bool es [f, l) un heap
accumulate	f, l, i, [op]	$T = \sum / \text{oper de } [f, l)$
inner_product	f1, l1, f2, i	$T = i + [f1, l1) \cdot [f2, \dots)$
partial_sum	f, l, r, [op]	$r+i = \sum / \text{oper de } [f, f+i) \forall i \in [f, l)$
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

2. Estructuras

2.1. RMQ (static)

Dado un arreglo y una operación asociativa idempotente, get(i, j) opera sobre el rango [i, j). Restricción: LVL $\geq 2 \cdot \text{ceil}(\log n)$; Usar [] para llenar arreglo y luego build().

```

1 struct RMQ{
2     #define LVL 10
3     tipo vec[LVL][1<<(LVL+1)];
4     tipo &operator[] (int p){return vec[0][p];}
5     tipo get(int i, int j) { //intervalo [i,j)
6         int p = 31-__builtin_clz(j-i);
7         return min(vec[p][i], vec[p][j-(1<<p)]);
8     }
9     void build(int n) { //O(n log n)
10        int mp = 31-__builtin_clz(n);
11        forn(p, mp) forn(x, n-(1<<p))
12            vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
13    }
14 };

```

2.2. RMQ (dynamic)

```

1 //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
  //sobre el rango [i, j).
2 #define MAXN 100000
3 struct RMQ{
4     int sz;
5     tipo t[4*MAXN];
6     tipo &operator[] (int p){return t[sz+p];}
7     void init(int n){ //O(n log n)
8         sz = 1 << (32-__builtin_clz(n));
9         fill(t, t+2*sz, 0); // 0=elemento neutro
10    }
11    void updall(){ //O(n)
12        dforsn(i, 0, sz) t[i]=max(t[2*i], t[2*i+1]);}
13    tipo get(int i, int j){return get(i, j, 1, 0, sz);}
14    tipo get(int i, int j, int n, int a, int b){ //O(lgn)
15        if(j<=a || i>=b) return 0; //neutro
16        if(i<=a && b<=j) return t[n];
17        int c=(a+b)/2;
18        return max(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));

```

```

19 }
20 void set(int p, tipo val){//0(lgn)
21     for(p+=sz; p>0 && t[p]!=val;){
22         t[p]=val;
23         p/=2;
24         val=max(t[p*2], t[p*2+1]);
25     }
26 }
27 }rmq;
28 //Usage:
29 cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();

```

2.3. Fenwick Tree

```

1 //For 2D threat each column as a Fenwick tree, by adding a nested for in
  //each operation
2 struct Fenwick{
3     static const int sz=1000001;
4     tipo t[sz];
5     tipo sum(int a, int b){return sum(b)-sum(a-1);}
6     void adjust(int p, tipo v){//valid with p in [1, sz), 0(lgn)
7         for(; p<sz; p+=(p&-p)) t[p]+=v; }
8     tipo sum(int p){//cumulative sum in [1, p], 0(lgn)
9         tipo s=0;
10        for(; p; p-=(p&-p)) s+=t[p];
11        return s;
12    }
13    //get largest value with cumulative sum less than or equal to x;
14    //for smallest, pass x-1 and add 1 to result
15    int getind(tipo x) {//0(lgn)
16        int idx = 0, mask = N;
17        while(mask && idx < N) {
18            int t = idx + mask;
19            if(x >= tree[t])
20                idx = t, x -= tree[t];
21            mask >>= 1;
22        }
23        return idx;
24    }
25 };

```

2.4. Union Find

```

1 struct UnionFind{

```

```

2     vector<int> f;//the array contains the parent of each node
3     void init(int n){f.clear(); f.insert(f.begin(), n, -1);}
4     int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));} //0(1)
5     bool join(int i, int j) {
6         bool con=comp(i)==comp(j);
7         if(!con) f[comp(i)] = comp(j);
8         return con;
9     }
10 };

```

2.5. Disjoint Intervals

```

1 bool operator< (const ii &a, const ii &b) {return a.fst<b.fst;}
2 //Stores intervals as [first, second]
3 //in case of a collision it joins them in a single interval
4 struct disjoint_intervals {
5     set<ii> segs;
6     void insert(ii v) {//0(lgn)
7         if(v.snd-v.fst==0.) return;//0J0
8         set<ii>::iterator it,at;
9         at = it = segs.lower_bound(v);
10        if (at!=segs.begin() && (--at)->snd >= v.fst)
11            v.fst = at->fst, --it;
12        for(; it!=segs.end() && it->fst <= v.snd; segs.erase(it++))
13            v.snd=max(v.snd, it->snd);
14        segs.insert(v);
15    }
16 };

```

2.6. RMQ (2D)

```

1 struct RMQ2D{
2     static const int sz=1024;
3     RMQ t[sz];
4     RMQ &operator[] (int p){return t[sz/2+p];}
5     void build(int n, int m){//0(nm)
6         forr(y, sz/2, sz/2+m)
7             t[y].build(m);
8         forr(y, sz/2+m, sz)
9             forn(x, sz)
10                t[y].t[x]=0;
11        dforn(y, sz/2)
12            forn(x, sz)
13                t[y].t[x]=max(t[y*2].t[x], t[y*2+1].t[x]);

```

```

14 }
15 void set(int x, int y, tipo v){//0(lgm.lgn)
16     y+=sz/2;
17     t[y].set(x, v);
18     while(y/=2)
19         t[y].set(x, max(t[y*2][x], t[y*2+1][x]));
20 }
21 //0(lgm.lgn)
22 int get(int x1, int y1, int x2, int y2, int n=1, int a=0, int b=sz/2){
23     if(y2<=a || y1>=b) return 0;
24     if(y1<=a && b<=y2) return t[n].get(x1, x2);
25     int c=(a+b)/2;
26     return max(get(x1, y1, x2, y2, 2*n, a, c),
27               get(x1, y1, x2, y2, 2*n+1, c, b));
28 }
29 };
30 //Example to initialize a grid of M rows and N columns:
31 RMQ2D rmq;
32 forn(i, M)
33     forn(j, N)
34         cin >> rmq[i][j];
35 rmq.build(N, M);

```

2.7. Big Int

```

1 #define BASEXP 6
2 #define BASE 1000000
3 #define LMAX 1000
4 struct bint{
5     int l;
6     tipo n[LMAX];
7     bint(tipo x){
8         l=0;
9         forn(i, LMAX){
10             n[i]=x%BASE;
11             x/=BASE;
12             l+=!!x||!i;
13         }
14     }
15     bint(){THIS = bint(0);}
16     bint(string x){
17         l=(x.size()-1)/BASEXP+1;
18         fill(n, n+LMAX, 0);

```

```

19     tipo r=1;
20     forn(i, sz(x)){
21         n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
22         r*=10; if(r==BASE)r=1;
23     }
24 }
25 void out(){
26     cout << n[l-1];
27     dfor(n, l-1) printf("%6.6llu", n[i]); //6=BASEXP!
28 }
29 void invar(){
30     fill(n+1, n+LMAX, 0);
31     while(l>1 && !n[l-1]) l--;
32 }
33 };
34 bint operator+(const bint&a, const bint&b){
35     bint c;
36     c.l = max(a.l, b.l);
37     tipo q = 0;
38     forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q %BASE, q/=BASE;
39     if(q) c.n[c.l++] = q;
40     c.invar();
41     return c;
42 }
43 pair<bint, bool> lresta(const bint& a, const bint& b) // c = a - b
44 {
45     bint c;
46     c.l = max(a.l, b.l);
47     tipo q = 0;
48     forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE) %BASE, q=(q+BASE)/
49         BASE-1;
50     c.invar();
51     return mkp(c, !q);
52 }
53 bint& operator-= (bint& a, const bint& b){return a=lresta(a, b).fst;}
54 bint operator- (const bint&a, const bint&b){return lresta(a, b).fst;}
55 bool operator< (const bint&a, const bint&b){return !lresta(a, b).snd;}
56 bool operator<= (const bint&a, const bint&b){return lresta(b, a).snd;}
57 bool operator==(const bint&a, const bint&b){return a <= b && b <= a;}
58 bint operator*(const bint&a, tipo b){
59     bint c;
60     tipo q = 0;
61     forn(i, a.l) q += a.n[i]*b, c.n[i] = q %BASE, q/=BASE;

```

```

61     c.l = a.l;
62     while(q) c.n[c.l++] = q %BASE, q/=BASE;
63     c.invar();
64     return c;
65 }
66 bint operator*(const bint&a, const bint&b){
67     bint c;
68     c.l = a.l+b.l;
69     fill(c.n, c.n+b.l, 0);
70     forn(i, a.l){
71         tipo q = 0;
72         forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q %BASE, q
73             /=BASE;
74         c.n[i+b.l] = q;
75     }
76     c.invar();
77     return c;
78 }
79 pair<bint, tipo> ldiv(const bint& a, tipo b){// c = a / b ; rm = a % b
80     bint c;
81     tipo rm = 0;
82     dforn(i, a.l){
83         rm = rm * BASE + a.n[i];
84         c.n[i] = rm / b;
85         rm %= b;
86     }
87     c.l = a.l;
88     c.invar();
89     return mkp(c, rm);
90 }
91 bint operator/(const bint&a, tipo b){return ldiv(a, b).fst;}
92 tipo operator%(const bint&a, tipo b){return ldiv(a, b).snd;}
93 pair<bint, bint> ldiv(const bint& a, const bint& b){
94     bint c;
95     bint rm = 0;
96     dforn(i, a.l){
97         if (rm.l==1 && !rm.n[0])
98             rm.n[0] = a.n[i];
99         else{
100             dforn(j, rm.l) rm.n[j+1] = rm.n[j];
101             rm.n[0] = a.n[i];
102             rm.l++;
103         }
104     }

```

```

103     tipo q = rm.n[b.l] * BASE + rm.n[b.l-1];
104     tipo u = q / (b.n[b.l-1] + 1);
105     tipo v = q / b.n[b.l-1] + 1;
106     while (u < v-1){
107         tipo m = (u+v)/2;
108         if (b*m <= rm) u = m;
109         else v = m;
110     }
111     c.n[i]=u;
112     rm-=b*u;
113 }
114 c.l=a.l;
115 c.invar();
116 return mkp(c, rm);
117 }
118 bint operator/(const bint&a, const bint&b){return ldiv(a, b).fst;}
119 bint operator%(const bint&a, const bint&b){return ldiv(a, b).snd;}

```

2.8. Modnum

```

1 struct mnum{
2     static const tipo mod=12582917;
3     tipo v;
4     mnum(tipo v=0): v(v%mod) {}
5     mnum operator+(mnum b){return v+b.v;}
6     mnum operator-(mnum b){return v>=b.v? v-b.v : mod-b.v+v;}
7     mnum operator*(mnum b){return v*b.v;}
8     mnum operator^(int n){
9         if(!n) return 1;
10        return n%2? (*this)^(n/2)*(*this) : (*this)^(n/2);}
11 };

```

2.9. Bittrie

```

1 struct bitrie{
2     static const int sz=1<<5;//5=ceil(log(n))
3     int V;//valor del nodo
4     vector<bitrie> ch;//childs
5     bitrie():V(0){} //NEUTRO
6     void set(int p, int v, int bit=sz>>1){//0(log sz)
7         if(bit){
8             ch.resize(2);
9             ch[(p&bit)>0].set(p, v, bit>>1);
10            V=max(ch[0].V, ch[1].V);

```

```

11     }
12     else V=v;
13 }
14 int get(int i, int j, int a=0, int b=sz){//0(log sz)
15     if(j<=a || i>=b) return 0;//NEUTRO
16     if(i<=a && b<=j) return V;
17     if(!sz(ch)) return V;
18     int c=(a+b)/2;
19     return max(ch[0].get(i, j, a, c), ch[1].get(i, j, c, b));
20 }
21 };

```

3. Strings

3.1. Trie

```

1 struct trie{
2     map<char, trie> m;
3     void add(const string &s, int p=0){
4         if(s[p]) m[s[p]].add(s, p+1);
5     }
6     void dfs(){
7         //Do stuff
8         forall(it, m)
9             it->second.dfs();
10    }
11 };

```

3.2. Suffix Array

```

1 #define MAX_N 1000
2 #define RABOUND(x) (x<n? RA[x] : 0)
3 //SA will hold the suffixes in order.
4 int SA[MAX_N], RA[MAX_N], n;
5 string s; //input string, n=sz(s)
6
7 void countingSort(int k){
8     int f[MAX_N], tmpSA[MAX_N];
9     zero(f);
10    forn(i, n) f[RABOUND(i+k)]++;
11    int sum=0;
12    forn(i, max(255, n)){
13        int t=f[i]; f[i]=sum; sum+=t;}

```

```

14    forn(i, n)
15        tmpSA[f[RABOUND(SA[i]+k)]++] = SA[i];
16    memcpy(SA, tmpSA, sizeof(SA));
17 }
18 void constructSA(){//0(n log n)
19     n=sz(s);
20     forn(i, n) SA[i]=i, RA[i]=s[i];
21     for(int k=1; k<n; k<=<1){
22         countingSort(k), countingSort(0);
23         int r, tmpRA[MAX_N];
24         tmpRA[SA[0]]=r=0;
25         forr(i, 1, n)
26             tmpRA[SA[i]]=(RA[SA[i]]==RA[SA[i-1]] && RA[SA[i]+k]==RA[SA[i-1]+k])? r : ++r;
27         memcpy(RA, tmpRA, sizeof(RA));
28         if(RA[SA[n-1]]==n-1) break;
29     }
30 }
31 void print(){//for debug
32     forn(i, n)
33         cout << i << ' ' <<
34         s.substr(SA[i], s.find( '$', SA[i])-SA[i]) << endl;}

```

3.3. String Matching With Suffix Array

```

1 //returns (lowerbound, upperbound) of the search
2 ii stringMatching(string P){ //0(sz(P)lgn)
3     int lo=0, hi=n-1, mid=lo;
4     while(lo<hi){
5         mid=(lo+hi)/2;
6         int res=s.compare(SA[mid], sz(P), P);
7         if(res>=0) hi=mid;
8         else lo=mid+1;
9     }
10    if(s.compare(SA[lo], sz(P), P)!=0) return ii(-1, -1);
11    ii ans; ans.fst=lo;
12    lo=0, hi=n-1, mid;
13    while(lo<hi){
14        mid=(lo+hi)/2;
15        int res=s.compare(SA[mid], sz(P), P);
16        if(res>0) hi=mid;
17        else lo=mid+1;
18    }

```

```

19 if(s.compare(SA[hi], sz(P), P)!=0) hi--;
20 ans.snd=hi;
21 return ans;
22 }

```

3.4. LCP (Longest Common Prefix)

```

1 //Calculates the LCP between consecutives suffixes in the Suffix Array.
2 //LCP[i] is the length of the LCP between SA[i] and SA[i-1]
3 int LCP[MAX_N];
4 void computeLCP(){//O(n)
5     int phi[MAX_N], PLCP[MAX_N];
6     phi[SA[0]]=-1;
7     forr(i, 1, n) phi[SA[i]]=SA[i-1];
8     int L=0;
9     forn(i, n){
10         if(phi[i]==-1) {PLCP[i]=0; continue;}
11         while(s[i+L]==s[phi[i]+L]) L++;
12         PLCP[i]=L;
13         L=max(L-1, 0);
14     }
15     forn(i, n) LCP[i]=PLCP[SA[i]];
16 }

```

3.5. Corasick

```

1
2 struct trie{
3     map<char, trie> next;
4     trie* tran[256];//transiciones del automata
5     int idhoja, szhoja;//id de la hoja o 0 si no lo es
6     //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que
7     //es hoja
8     trie *padre, *link, *nxthoja;
9     char pch;//caracter que conecta con padre
10    trie(): tran(), idhoja(), padre(), link() {}
11    void insert(const string &s, int id=1, int p=0){//id>0!!!
12        if(p<sz(s)){
13            trie &ch=next[s[p]];
14            tran[(int)s[p]]=&ch;
15            ch.padre=this, ch.pch=s[p];
16            ch.insert(s, id, p+1);
17        }
18        else idhoja=id, szhoja=sz(s);
19    }

```

```

18 }
19 trie* get_link() {
20     if(!link){
21         if(!padre) link=this;//es la raiz
22         else if(!padre->padre) link=padre;//hijo de la raiz
23         else link=padre->get_link()->get_tran(pch);
24     }
25     return link;
26 }
27 trie* get_tran(int c) {
28     if(!tran[c])
29         tran[c] = !padre? this : this->get_link()->get_tran(c);
30     return tran[c];
31 }
32 trie *get_nxthoja(){
33     if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
34     return nxthoja;
35 }
36 void print(int p){
37     if(idhoja)
38         cout << "found_" << idhoja << "_at_position_" << p-szhoja << endl
39         ;
40     if(get_nxthoja()) get_nxthoja()->print(p);
41 }
42 void matching(const string &s, int p=0){
43     print(p);
44     if(p<sz(s)) get_tran(s[p])->matching(s, p+1);
45 }

```

4. Geometría

#define EPS 1e-9

4.1. Punto

```

1 struct pto{
2     tipo x, y;
3     pto(tipo x=0, tipo y=0):x(x),y(y){}
4     pto operator+(pto a){return pto(x+a.x, y+a.y);}
5     pto operator-(pto a){return pto(x-a.x, y-a.y);}
6     pto operator+(tipo a){return pto(x+a, y+a);}
7     pto operator*(tipo a){return pto(x*a, y*a);}
8     pto operator/(tipo a){return pto(x/a, y/a);}
9     //dot product, producto interno:

```



```

10  tipo operator*(pto a){return x*a.x+y*a.y;}
11  //module of the cross product or vectorial product:
12  //if a is less than 180 clockwise from b, a^b>0
13  tipo operator^(pto a){return x*a.y-y*a.x;}
14  //returns true if this is at the left side of line qr
15  bool left(pto q, pto r){return ((q-*this)^(r-*this))>0;}
16  bool operator<(const pto &a) const{return x<a.x || (abs(x-a.x)<EPS &&
    y<a.y);}
17  bool operator==(pto a){return abs(x-a.x)<EPS && abs(y-a.y)<EPS;}
18  double norm(){return sqrt(x*x+y*y);}
19  tipo norm_sq(){return x*x+y*y;}
20  };
21  double dist(pto a, pto b){return (b-a).norm();}
22  typedef pto vec;
23
24  double angle(pto a, pto o, pto b){
25      vec oa=a-o, ob=b-o;
26      return acos((oa*ob) / sqrt(oa.norm_sq()*ob.norm_sq()));}
27
28  //rotate p by theta rads CCW w.r.t. origin (0,0)
29  pto rotate(pto p, double theta){
30      return pto(p.x*cos(theta)-p.y*sin(theta),
31          p.x*sin(theta)+p.y*cos(theta));
32  }

```

4.2. Line

```

1  struct line{
2      line() {}
3      double a,b,c;//Ax+By=C
4      //pto MUST store float coordinates!
5      line(double a, double b, double c):a(a),b(b),c(c){}
6      line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
7  };
8  bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
9  pto inter(line l1, line l2){//intersection
10     double det=l1.a*l2.b-l2.a*l1.b;
11     if(abs(det)<EPS) return pto(INF, INF);//parallels
12     return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
13 }

```

4.3. Segment

```

1  struct segm{

```

```

2      pto s,f;
3      segm(pto s, pto f):s(s), f(f) {}
4      pto closest(pto p) {//use for dist to point
5          double l2 = dist_sq(s, f);
6          if(l2==0.) return s;
7          double t =((p-s)*(f-s))/l2;
8          if (t<0.) return s;//not write if is a line
9          else if(t>1.)return f;//not write if is a line
10         return s+((f-s)*t);
11     }
12     bool inside(pto p){
13 return ((s-p)^(f-p))==0 && min(s, f)<*this&&*this<max(s, f);}
14 };
15
16 bool insidebox(pto a, pto b, pto p) {
17     return (a.x-p.x)*(p.x-b.x)>-EPS && (a.y-p.y)*(p.y-b.y)>-EPS;
18 }
19 pto inter(segm s1, segm s2){
20     pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
21     if(insidebox(s1.s,s1.f,p) && insidebox(s2.s,s2.f,p))
22         return r;
23     return pto(INF, INF);
24 }

```

4.4. Rectangle

```

1  struct rect{
2      //lower-left and upper-right corners
3      pto lw, up;
4  };
5  //returns if there's an intersection and stores it in r
6  bool inter(rect a, rect b, rect &r){
7      r.lw=pto(max(a.lw.x, b.lw.x), max(a.lw.y, b.lw.y));
8      r.up=pto(min(a.up.x, b.up.x), min(a.up.y, b.up.y));
9      //check case when only a edge is common
10     return r.lw.x<r.up.x && r.lw.y<r.up.y;
11 }

```

4.5. Polygon Area

```

1  double area(vector<tipo> &p){//0(sz(p))
2      double area=0;
3      for(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
4      //if points are in clockwise order then area is negative

```



```

5   return abs(area)/2;
6 }
7 //Area ellipse = M_PI*a*b where a and b are the semi axis lengths
8 //Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2

```

4.6. Circle

```

1 vec perp(vec v){return vec(-v.y, v.x);}
2 line bisector(pto x, pto y){
3     line l=line(x, y); pto m=(x+y)/2;
4     return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
5 }
6 struct Circle{
7     pto o;
8     double r;
9     //circle determined by three points, uses line
10    Circle(pto x, pto y, pto z){
11        o=inter(bisector(x, y), bisector(y, z));
12        r=dist(o, x);
13    }
14    pair<pto, pto> ptosTang(pto p){
15        pto m=(p+o)/2;
16        tipo d=dist(o, m);
17        tipo a=r*r/(2*d);
18        tipo h=sqrt(r*r-a*a);
19        pto m2=o+(m-o)*a/d;
20        vec per=perp(m-o)/d;
21        return mkp(m2-per*h, m2+per*h);
22    }
23 };
24 //finds the center of the circle containing p1 and p2 with radius r
25 //as there may be two solutions swap p1, p2 to get the other
26 bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
27     double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
28     if(det<0) return false;
29     c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
30     return true;
31 }

```

4.7. Point in Poly

```

1 //checks if v is inside of P, using ray casting
2 //works with convex and concave.
3 //excludes boundaries, handle it separately using segment.inside()

```

```

4 bool inPolygon(pto v, vector<pto>& P) {
5     bool c = false;
6     forn(i, sz(P)){
7         int j=(i+1)%sz(P);
8         if((P[j].y>v.y) != (P[i].y > v.y) &&
9         (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
10            c = !c;
11     }
12     return c;
13 }

```

4.8. Convex Check CHECK

```

1 bool isConvex(vector<int> &p){//O(N)
2     int N=sz(p);
3     if(N<3) return false;
4     bool isLeft=p[0].left(p[1], p[2]);
5     forr(i, 1, N)
6         if(p[i].left(p[(i+1)%N], p[(i+2)%N])!=isLeft)
7             return false;
8     return true; }

```

4.9. Convex Hull

```

1 //stores convex hull of P in S, CCW order
2 void CH(vector<pto>& P, vector<pto> &S){
3     S.clear();
4     sort(P.begin(), P.end());
5     forn(i, sz(P)){
6         while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
7         S.pb(P[i]);
8     }
9     S.pop_back();
10    int k=sz(S);
11    dforn(i, sz(P)){
12        while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back
13            ();
14        S.pb(P[i]);
15    }
16    S.pop_back();
17 }

```

4.10. Cut Polygon

```

1 //cuts polygon Q along the line ab
2 //stores the left side (swap a, b for the right one) in P
3 void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
4     P.clear();
5     forn(i, sz(Q)){
6         double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
7         if(left1>=0) P.pb(Q[i]);
8         if(left1*left2<0)
9             P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
10    }
11 }

```

4.11. Bresenham

```

1 //plot a line approximation in a 2d map
2 void bresenham(pto a, pto b){
3     pto d=b-a; d.x=abs(d.x), d.y=abs(d.y);
4     pto s(a.x<b.x? 1: -1, a.y<b.y? 1: -1);
5     int err=d.x-d.y;
6     while(1){
7         m[a.x][a.y]=1;//plot
8         if(a==b) break;
9         int e2=2*err;
10        if(e2 > -d.y){
11            err-=d.y, a.x+=s.x;
12        }
13        if(e2 < d.x)
14            err+= d.x, a.y+= s.y;
15    }
16 }

```

4.12. Rotate Matrix

```

1 //rotates matrix t 90 degrees clockwise
2 //using auxiliary matrix t2(faster)
3 void rotate(){
4     forn(x, n) forn(y, n)
5         t2[n-y-1][x]=t[x][y];
6     memcpy(t, t2, sizeof(t));
7 }

```

5. Math

5.1. GCD

```

1 tipo gcd(tipo a, tipo b){return a?gcd(b %a, a):b;}

```

5.2. LCM

```

1 tipo lcm(tipo a, tipo b){return a / gcd(a,b) * b;}

```

5.3. Simpson

```

1 double integral(double a, double b, int n=10000) {//0(n), n=cantdiv
2     double area=0, h=(b-a)/n, fa=f(a), fb;
3     forn(i, n){
4         fb=f(a+h*(i+1));
5         area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
6     }
7     return area*h/6.;}

```

5.4. Fraction

```

1 struct frac{
2     tipo p,q;
3     frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
4     tipo mcd(tipo a, tipo b){return a?mcd(b %a, a):b;}
5     void norm(){
6         tipo a = mcd(p,q);
7         if(a) p/=a, q/=a;
8         else q=1;
9         if (q<0) q=-q, p=-p;}
10    frac operator+(const frac& o){
11        tipo a = mcd(q,o.q);
12        return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
13    frac operator-(const frac& o){
14        tipo a = mcd(q,o.q);
15        return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
16    frac operator*(frac o){
17        tipo a = mcd(q,o.q), b = mcd(o.q,p);
18        return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
19    frac operator/(frac o){
20        tipo a = mcd(q,o.q), b = mcd(o.p,p);
21        return frac((p/b)*(o.q/a), (q/a)*(o.p/b));}
22    bool operator<(frac o){return (*this-o).p<0;}
23    bool operator==(frac o){return p==o.p&&q==o.q;}
24 };

```

5.5. Polinomio

```

1 #define MAX_GR 20
2 struct poly {
3     tipo p[MAX_GR]; //guarda los coeficientes del polinomio
4     poly(){zero(p);}
5     int gr(){//calculates grade of the polynomial
6         dforn(i,MAX_GR) if(p[i]) return i;
7         return 0; }
8     bool isnull() {return gr()==0 && !p[0];}
9     poly operator+(poly b) {// - is analogous
10         poly c=THIS;
11         forn(i,MAX_GR) c.p[i]+=b.p[i];
12         return c;
13     }
14     poly operator*(poly b) {
15         poly c;
16         forn(i,MAX_GR) forn(k,i+1) c.p[i]+=p[k]*b.p[i-k];
17         return c;
18     }
19     tipo eval(tipo v) {
20         tipo sum = 0;
21         dforsn(i, 0, MAX_GR) sum=sum*v + p[i];
22         return sum;
23     }
24     //the following function generates the roots of the polynomial
25     //it can be easily modified to return float roots
26     set<tipo> roots(){
27         set<tipo> roots;
28         tipo a0 = abs(p[0]), an = abs(p[gr()]);
29         vector<tipo> ps,qs;
30         forr(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps.pb(a0/p);
31         forr(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs.pb(an/q);
32         forall(pt,ps)
33             forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
34                 tipo root = abs((*pt) / (*qt));
35                 if (eval(root)==0) roots.insert(root);
36             }
37         return roots;
38     }
39 };
40 //the following functions allows parsing an expression like
41 //34+150+4*45
42 //into a polynomial(el numero en funcion de la base)
43 #define LAST(s) (sz(s)? s[sz(s)-1] : 0)

```

```

44 #define POP(s) s.erase(--s.end());
45 poly D(string &s) {
46     poly d;
47     for(int i=0; isdigit(LAST(s)); i++) d.p[i]=LAST(s)-'0', POP(s);
48     return d;}
49
50 poly T(string &s) {
51     poly t=D(s);
52     if (LAST(s)=='*'){POP(s); return T(s)*t;}
53     return t;
54 }
55 //main function, call this to parse
56 poly E(string &s) {
57     poly e=T(s);
58     if (LAST(s)=='+'){POP(s); return E(s)+e;}
59     return e;
60 }

```

6. Grafos

6.1. Dijkstra

```

1 #define INF 1e9
2 int N;
3 #define MAX_V 250001
4 vector<ii> G[MAX_V];
5 //To add an edge use
6 #define add(a, b, w) G[a].pb(mkp(w, b))
7
8 ll dijkstra(int s, int t){//O(|E| log |V|)
9     priority_queue<ii, vector<ii>, greater<ii> > Q;
10     vector<ll> dist(N, INF); vector<int> dad(N, -1);
11     Q.push(mkp(0, s)); dist[s] = 0;
12     while(sz(Q)){
13         ii p = Q.top(); Q.pop();
14         if(p.snd == t) break;
15         forall(it, G[p.snd])
16             if(dist[p.snd]+it->first < dist[it->snd]){
17                 dist[it->snd] = dist[p.snd] + it->fst;
18                 dad[it->snd] = p.snd;
19                 Q.push(mkp(dist[it->snd], it->snd));
20             }
21     }

```

```

22 return dist[t];
23 if(dist[t]<INF)//path generator
24     for(int i=t; i!=-1; i=dad[i])
25         printf("%d%c", i, (i==s?'\\n':'\\u'));
26 }

```

6.2. Bellman-Ford

```

1 vector<ii> G[MAX_N]; //ady. list with pairs (weight, dst)
2 int dist[MAX_N];
3 void bford(int src){ //O(VE)
4     dist[src]=0;
5     forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
6         dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
7 }
8
9 bool hasNegCycle(){
10     forn(j, N) if(dist[j]!=INF) forall(it, G[j])
11         if(dist[it->snd]>dist[j]+it->fst) return true;
12     //inside if: all points reachable from it->snd will have -INF distance
13     // (do bfs)
14     return false;
15 }

```

6.3. Floyd-Warshall

```

1 //G[i][j] contains weight of edge (i, j) or INF
2 //G[i][i]=0
3 int G[MAX_N][MAX_N];
4 void floyd(){ //O(N^3)
5     forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
6         G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
7 }
8 bool inNegCycle(int v){
9     return G[v][v]<0;
10 }
11 //checks if there's a neg. cycle in path from a to b
12 bool hasNegCycle(int a, int b){
13     forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
14         return true;
15     return false;
16 }

```

6.4. 2-SAT + Tarjan SCC

```

1 //We have a vertex representing a var and other for his negation.
2 //Every edge stored in G represents an implication. To add an equation
3 //of the form a||b, use addor(a, b)
4 //MAX=max cant var, n=cant var
5 #define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
6 vector<int> G[MAX*2];
7 //idx[i]=index assigned in the dfs
8 //lw[i]=lowest index(closer from the root) reachable from i
9 int lw[MAX*2], idx[MAX*2], qidx;
10 stack<int> q;
11 int qcmp, cmp[MAX*2];
12 //verdad[cmp[i]]=valor de la variable i
13 bool verdad[MAX*2+1];
14
15 int neg(int x) { return x>=n? x-n : x+n; }
16 void tjn(int v){
17     lw[v]=idx[v]=++qidx;
18     q.push(v), cmp[v]=-2;
19     forall(it, G[v]){
20         if(!idx[*it] || cmp[*it]==-2){
21             if(!idx[*it]) tjn(*it);
22             lw[v]=min(lw[v], lw[*it]);
23         }
24     }
25     if(lw[v]==idx[v]){
26         qcmp++;
27         int x;
28         do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
29         verdad[qcmp]=(cmp[neg(v)]<0);
30     }
31 }
32 //remember to CLEAR G!!!
33 bool satisf(){ //O(n)
34     memset(idx, 0, sizeof(idx)), qidx=0;
35     memset(cmp, -1, sizeof(cmp)), qcmp=0;
36     forn(i, n){
37         if(!idx[i]) tjn(i);
38         if(!idx[neg(i)]) tjn(neg(i));
39     }
40     forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
41     return true;
42 }

```

6.5. Articulation Points

```

1 int N;
2 vector<int> G[1000000];
3 //V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
4 int qV, V[1000000], L[1000000], P[1000000];
5 void dfs(int v, int f){
6     L[v]=V[v]=++qV;
7     forall(it, G[v])
8         if(!V[*it]){
9             dfs(*it, v);
10            L[v] = min(L[v], L[*it]);
11            P[v] += L[*it] >= V[v];
12        }
13        else if(*it != f)
14            L[v] = min(L[v], V[*it]);
15 }
16 int cantart(){ //O(n)
17     qV=0;
18     zero(V), zero(P);
19     dfs(1, 0); P[1]--;
20     int q=0;
21     forn(i, N) if(P[i]) q++;
22     return q;
23 }

```

6.6. LCA + Climb

7. Network Flow

7.1. Edmonds Karp's

```

1 #define MAX_V 1000
2 #define INF 1e9
3 //special nodes
4 #define SRC 0
5 #define SNK 1
6 map<int, int> G[MAX_V]; //limpiar esto
7 //To add an edge use
8 #define add(a, b, w) G[a][b]=w
9 int f, p[MAX_V];
10 void augment(int v, int minE){
11     if(v==SRC) f=minE;

```

```

12     else if(p[v] != -1){
13         augment(p[v], min(minE, G[p[v]][v]));
14         G[p[v]][v] -= f, G[v][p[v]] += f;
15     }
16 }
17 ll maxflow(){ //O(VE^2)
18     ll Mf=0;
19     do{
20         f=0;
21         char used[MAX_V]; queue<int> q; q.push(SRC);
22         zero(used), memset(p, -1, sizeof(p));
23         while(sz(q)){
24             int u=q.front(); q.pop();
25             if(u==SNK) break;
26             forall(it, G[u])
27                 if(it->snd>0 && !used[it->fst])
28                     used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
29         }
30         augment(SNK, INF);
31         Mf+=f;
32     }while(f);
33     return Mf;
34 }

```

7.2. Push-Relabel

```

1 #define MAX_V 1000
2 int N; //valid nodes are [0...N-1]
3 #define INF 1e9
4 //special nodes
5 #define SRC 0
6 #define SNK 1
7 map<int, int> G[MAX_V];
8 //To add an edge use
9 #define add(a, b, w) G[a][b]=w
10 ll excess[MAX_V];
11 int height[MAX_V], active[MAX_V], count[2*MAX_V+1];
12 queue<int> Q;
13 void enqueue(int v) {
14     if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v); }
15 void push(int a, int b) {
16     int amt = min(excess[a], ll(G[a][b]));
17     if(height[a] <= height[b] || amt == 0) return;

```

```

18  G[a][b] -= amt, G[b][a] += amt;
19  excess[b] += amt, excess[a] -= amt;
20  enqueue(b);
21  }
22  void gap(int k) {
23      forn(v, N){
24          if (height[v] < k) continue;
25          count[height[v]]--;
26          height[v] = max(height[v], N+1);
27          count[height[v]]++;
28          enqueue(v);
29      }
30  }
31  void relabel(int v) {
32      count[height[v]]--;
33      height[v] = 2*N;
34      forall(it, G[v])
35          if(it->snd)
36              height[v] = min(height[v], height[it->fst] + 1);
37      count[height[v]]++;
38      enqueue(v);
39  }
40  ll maxflow() { // O(V^3)
41      zero(height), zero(active), zero(count), zero(excess);
42      count[0] = N-1;
43      count[N] = 1;
44      height[SRC] = N;
45      active[SRC] = active[SNK] = true;
46      forall(it, G[SRC]){
47          excess[SRC] += it->snd;
48          push(SRC, it->fst);
49      }
50      while(sz(Q)) {
51          int v = Q.front(); Q.pop();
52          active[v] = false;
53          forall(it, G[v]) push(v, it->fst);
54          if(excess[v] > 0)
55              count[height[v]] == 1? gap(height[v]):relabel(v);
56      }
57      ll mf=0;
58      forall(it, G[SRC]) mf+=G[it->fst][SRC];
59      return mf;
60  }

```

8. Ayudamemoria

Límites

```
1 #include <climits> //INT_MIN, LONG_MAX, ULLONG_MAX, etc.
```

Cant. decimales

```
1 #include <iomanip>
2 cout << setprecision(2) << fixed;
```

Rellenar con espacios(para justificar)

```
1 #include <iomanip>
2 cout << setfill('␣') << setw(3) << 2 << endl;
```

Leer hasta fin de línea

```
1 #include <sstream>
2 //hacer cin.ignore() antes de getline()
3 while(getline(cin, line)){
4     istringstream is(line);
5     while(is >> X)
6         cout << X << "␣";
7     cout << endl;
8 }
```

Aleatorios

```
1 #define RAND(a, b) (rand()%(b-a+1)+a)
2 srand(time(NULL));
```

Doubles Comp.

```
1 const double EPS = 1e-9;
2 x == y <=> fabs(x-y) < EPS
3 x > y <=> x > y + EPS
4 x >= y <=> x > y - EPS
```

Límites

```
1 #include <limits>
2 numeric_limits<T>
3     ::max()
4     ::min()
5     ::epsilon()
```

Muahaha

```
1 | #include <signal.h>
2 | void divzero(int p){
3 |     while(true);}
4 | void segm(int p){
5 |     exit(0);}
6 | //in main
7 | signal(SIGFPE, divzero);
8 | signal(SIGSEGV, segm);
```

Mejorar velocidad

```
1 | ios::sync_with_stdio(false);
```

Leer del teclado

```
1 | freopen("/dev/tty", "a", stdin);
```

File setup

```
1 | //tambien se pueden usar comas: {a, x, m, l}
2 | for i in {a..k}; do cp template.cpp $i.cpp; touch $i.in; done
```